OO Verification Research Results

John Joseph Chilenski
Associate Technical Fellow
Boeing Commercial Airplanes

July 27, 2005

Background

- Boeing is conducting a three phase research project into the verification of object-oriented technology (OOT)
  - Phase I was a survey of current OOT verification practices in use within commercial aviation projects
    - – Results were presented as part of the “OO ?” discussion earlier today
  - Phase II is an investigation into the data coupling and control coupling (DC3) aspects of OOT
    - – Results to date will be presented in this discussion

Background (continued)

- Phase III will be
  - An investigation into the adequacy of structural coverage analysis done at the object code level instead of the source code level in OOT
  - Identification of concerns and open issues concerning OOT software verification that identify issues requiring further research

Agenda

- Coupling = Dependence
- Object Oriented Issues
  - Inheritance
  - Aggregation
  - Association
  - Polymorphism
    - – Static Dispatch
    - – Dynamic Dispatch

Coupling = Dependence

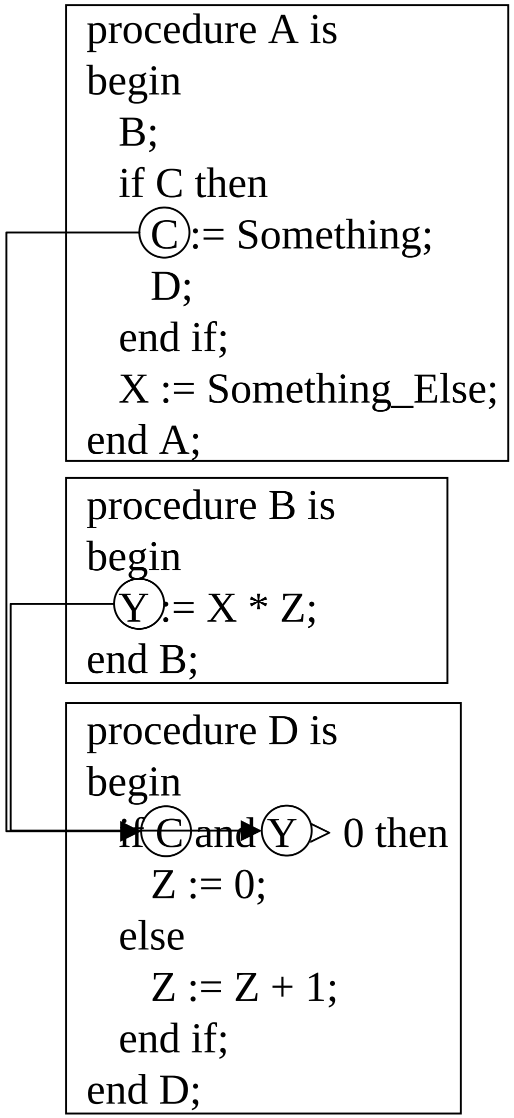
- Given the material in DO-178B, DO-248B FAQ#9 & FAQ#67 and CAST-19, we can conclude that the intent of the structural coverage analyses of the confirmation of DC3 is to:
  - Provide an objective assessment (measure) of the completeness of the requirements-based tests of the integrated components
    - Demonstration of the presence of intended interactions (function) between those components
  - Support the demonstration of the absence of unintended interactions (function) between those components
- This indicates that the confirmation of DC3 is specifically targeted at the integration process and its tests

Coupling = Dependence

- Integration focuses on dependencies and interfaces between components
- Semantic dependence between two program points has been shown to be uncomputable in the general case
- In standard CS usage, multiple components can be
  - Independent (uncoupled)
  - Dependent (coupled)
    - – Control Dependent
      - – control coupled
    - – Data Dependent
      - – data coupled
    - – Both
- Control and data dependence have been shown to be conservative approximations of semantic dependence

Coupling = Dependence

- In standard CS usage, a **data dependence** exists between two components if one component defines a data object and the other component uses that definition of the data object under some operational scenario
  - The data user is dependent on the data definer
    - – D is data dependent on A because of C
    - – D is data dependent on B because of Y



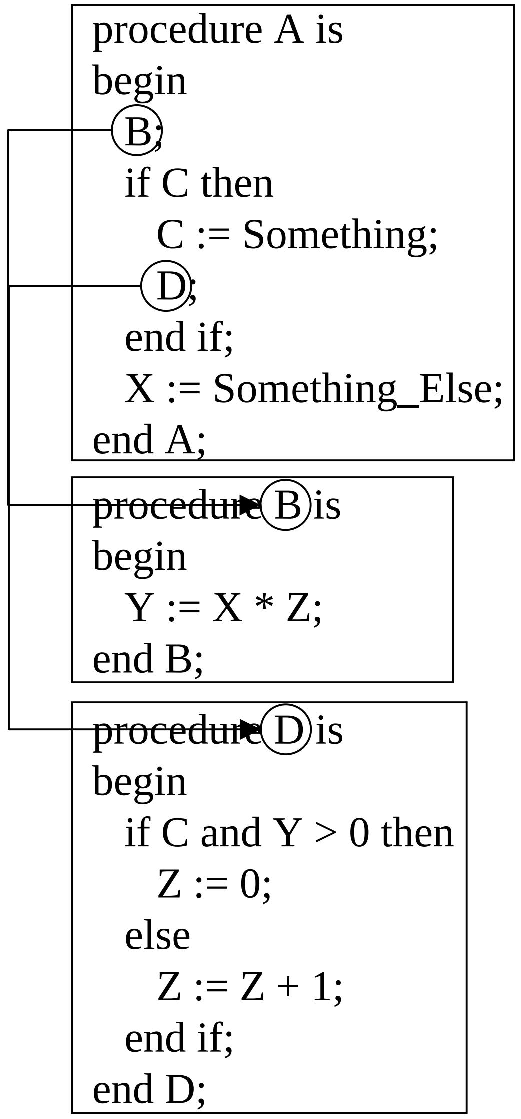
```
procedure A is
begin
   B;
   if C then
      C := Something;
      D;
   end if;
   X := Something_Else;
end A;
```

```
procedure B is
begin
   Y := X * Z;
end B;
```

```
procedure D is
begin
   if C and Y > 0 then
      Z := 0;
   else
      Z := Z + 1;
   end if;
end D;
```

Coupling = Dependence

- In standard CS usage, a **control dependence** exists between two components when the execution of one component depends on the other
  - One component calls the other under some operational scenario
    - – The callee is dependent on the caller
      - – B is control dependent on A because A calls B
      - – D is control dependent on A because A **conditionally** calls D



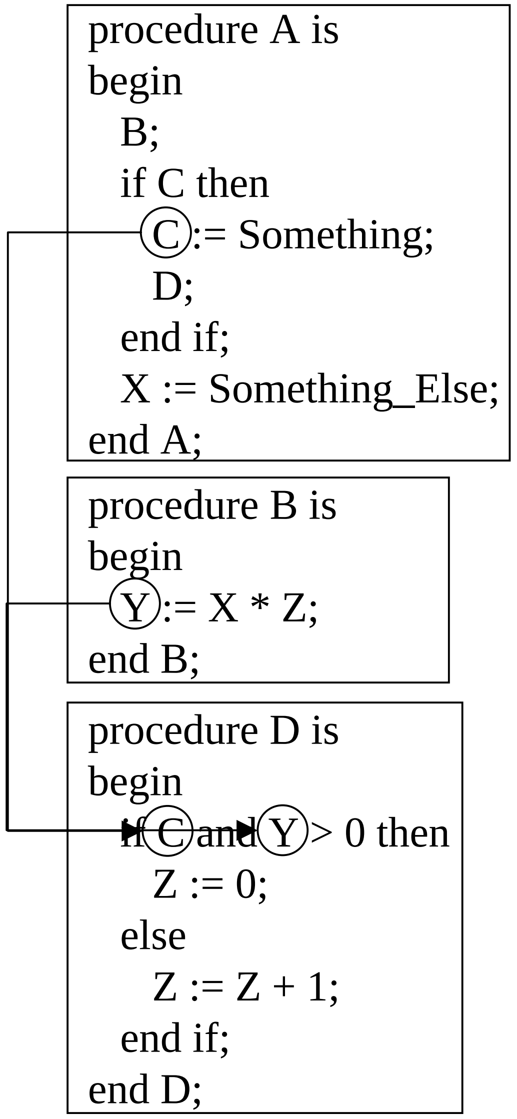
```
procedure A is
begin
   B;
   if C then
      C := Something;
      D;
   end if;
   X := Something_Else;
end A;
```

```
procedure B is
begin
   Y := X * Z;
end B;
```

```
procedure D is
begin
   if C and Y > 0 then
      Z := 0;
   else
      Z := Z + 1;
   end if;
end D;
```

Coupling = Dependence

- One component defines the data objects that determine the execution sequence taken by the other component under some operational scenario
  - This is just a special form of **data dependence** where the use of the data object is in a decision that determines whether the callee is called or not
    - D is control dependent on A because of C
    - D is control dependent on B because of Y



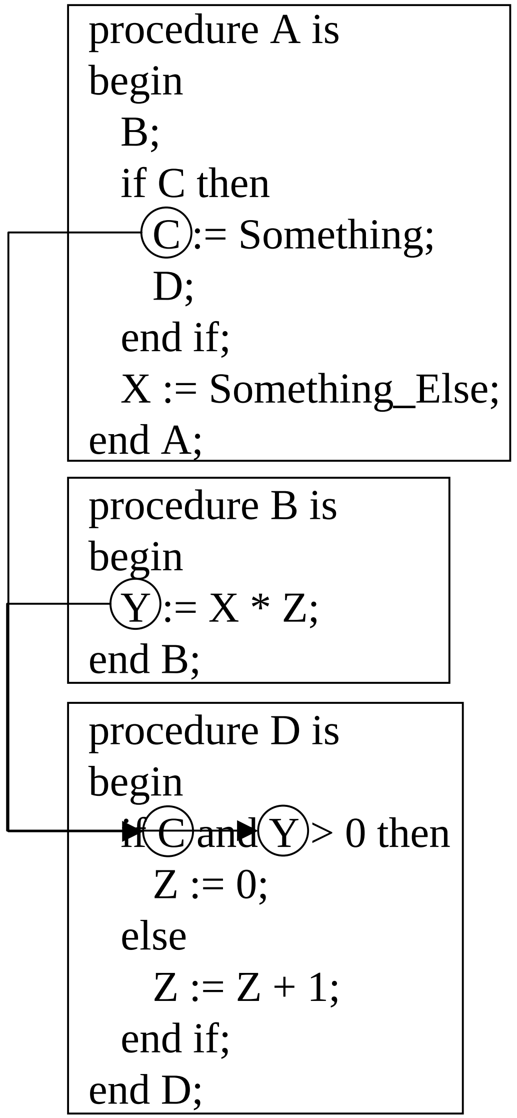
```
procedure A is
begin
   B;
   if C then
      C := Something;
      D;
   end if;
   X := Something_Else;
end A;
```

```
procedure B is
begin
   Y := X * Z;
end B;
```

```
procedure D is
begin
   if C and Y > 0 then
      Z := 0;
   else
      Z := Z + 1;
   end if;
end D;
```

Coupling = Dependence

- Verification of a data dependence can be accomplished by execution of a definition-use-association (DUA)
  - A DUA for an object X (d_X, u_X, X) is formed by a pair of statements:
    - – A definition statement (d_X) where X is given a value
    - – A use statement (u_X) where the value given to X in d_X is used
  - (A.5, D.3, C)
  - (B.3, D.3, Y)



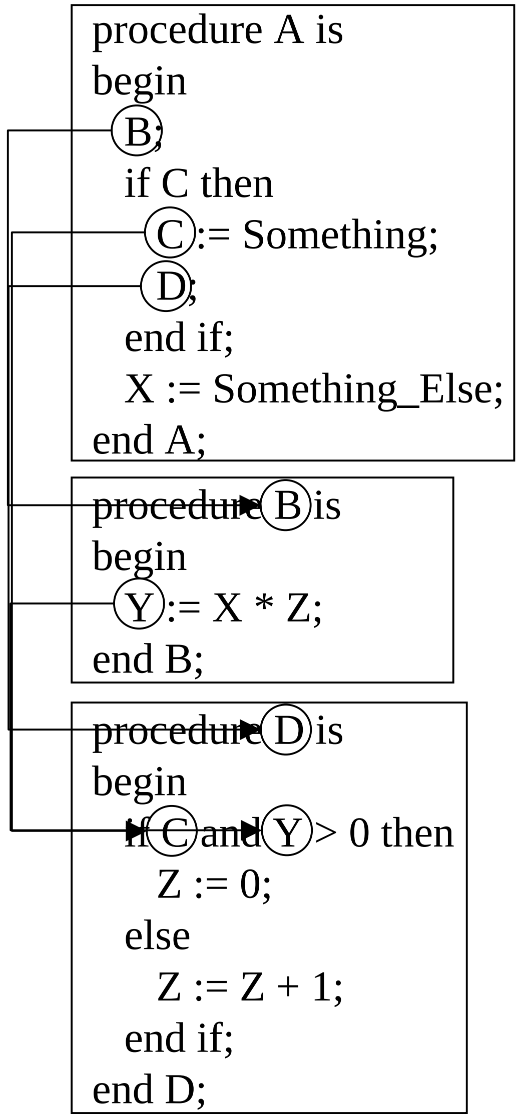
```
procedure A is
begin
   B;
   if C then
      C := Something;
      D;
   end if;
   X := Something_Else;
end A;
```

```
procedure B is
begin
   Y := X * Z;
end B;
```

```
procedure D is
begin
   if C and Y > 0 then
      Z := 0;
   else
      Z := Z + 1;
   end if;
end D;
```

Coupling = Dependence

- Verification of a control dependence can either be accomplished by execution of a DUA or call-association
  - A call-association between two components A, B, (A, c, B, p) is formed by
    - A call site c (a statement in A where B is called), and
    - A predicate p that identifies the conditions under which the call will occur
  - (A, A.3, B, A.entered)
  - (A, A.6, D, A.entered and C.A.4)
  - (A.5, D.3, C)
  - (B.3, D.3, Y)

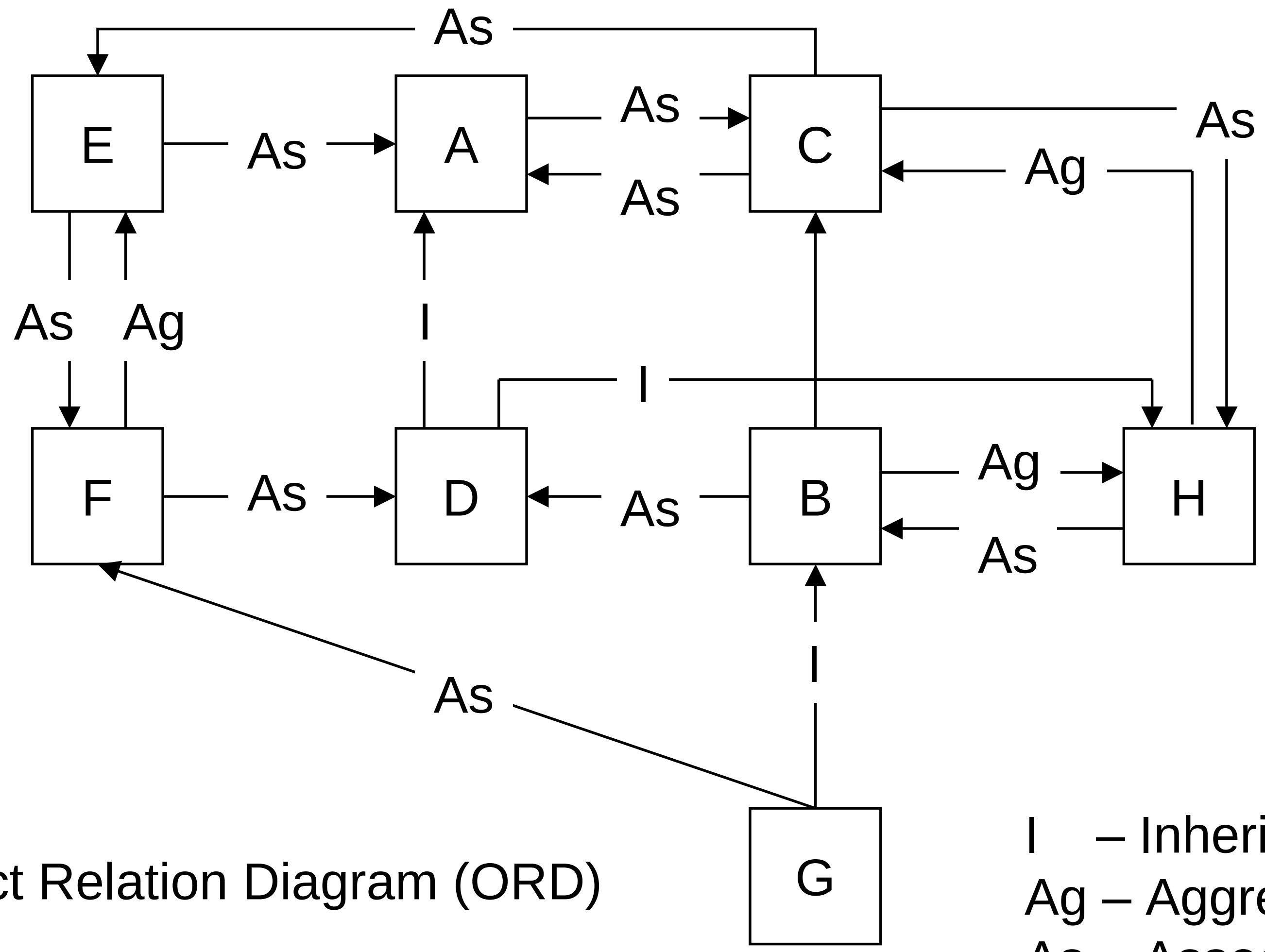


```
procedure A is
begin
   B;
   if C then
      C := Something;
      D;
   end if;
   X := Something_Else;
end A;

procedure B is
begin
   Y := X * Z;
end B;

procedure D is
begin
   if C and Y > 0 then
      Z := 0;
   else
      Z := Z + 1;
   end if;
end D;
```

Coupling = Dependence

- These analyses are standard in compiler optimization
- Coverage of DUA’s has been looked at for over 20 years
  - Data flow coverage
- Coverage of **inter-procedural/inter-class** DUA’s has been looked at as an integration testing adequacy criterion for over 15 years (inter-procedural) and is emerging for OOT (inter-class)
- Commercial tools are becoming available to perform these analyses
  - Including the coverage analysis
- Coverage of call associations requires further work

Object Oriented Issues – Inheritance, Aggregation, Association



Object Relation Diagram (ORD)

I – Inheritance
Ag – Aggregation
As – Association

Object Oriented Issues – Inheritance

- The parent class(es) should be tested before the child class
  - The hierarchical integration testing (HIT) methodology can be used to determine which parts of the parent need to be tested before which parts of the child
  - This is the last relationship which should be stubbed
    - These are the most complex stubs

Object Oriented Issues – Aggregation

- Objects of one class incorporate objects of other class(es) as attributes
- The encapsulated class should be tested before the encapsulating class
  - Only in the case of circular dependencies will stubs be needed

Object Oriented Issues – Association

- Call – one of A’s methods calls one of B’s methods
- Access – one of A’s methods accesses one of B’s attributes
- Parameter – one of A’s methods contains a parameter of type B

- The called class should be tested before the calling class
  - Only in the case of circular dependencies will stubs be needed
    - Apparently quite common

- In an ORD, this is considered the weakest form of dependency
  - This relationship should be broken/stubbed first
    - Least complex stubs
  - Many different weighting functions have been published
    - Break the one which requires the fewest stubs

Object Oriented Issues – Polymorphism – Static Dispatch

- With static dispatch, each reference resolves to a single entity (object or method)
  - This is what we are used to in procedural / imperative programming
- Each call site resolves to a single call association
- Only a single set of DUA’s exist
- They can all be tested as usual

Object Oriented Issues – Polymorphism – Dynamic Dispatch

- With dynamic dispatch, each reference resolves to a set of possible entities (objects or methods)
  - This is the famous pointer problem in C/C++
- Each call site resolves to a set of possible call associations
- Multiple sets of DUA’s exist
  - One for each possible call association
- Adequate testing of polymorphism is an active research area
  - No definitive answer yet

Object Oriented Issues – Polymorphism – Dynamic Dispatch

- Multiple approaches have been suggested
  - Every dispatch site has been executed and every possible dispatch target has been executed
    - Every possible object binding and every concrete method
    - Every possible object/method binding
      - “Flattened class” methods / dispatch table
      - Recommendation in the OOTiA Handbook
  - For each dispatch equivalence class, every dispatch site has been executed and every possible dispatch target has been executed

Object Oriented Issues – Polymorphism – Dynamic Dispatch

- For each dispatch equivalence class, every dispatch site has been executed and every dispatch target has been executed from at least one of those sites
- Every possible dispatch target has been executed from every dispatch site
  - Every possible object binding and every concrete method
  - Every possible object/method binding
    - Researchers agree this is probably “safe”
      - From the CS perspective
    - Researchers agree this is generally intractable

Conclusion

- Much work left to do
- Report due out before the end of the year
  - May not have the polymorphism problem solved
- Stay tuned …